BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Kedudukan selama melakukan proses kerja magang dalam perusahaan PT Global Tiket Network pada tim *Payment* subdivisi *Tech Platform* dibawah divisi *Technology* adalah sebagai *Frontend Engineer Intern.* Tim *Payment* dipimpin oleh *Technical Lead* yaitu Bapak Dekadwinavinta Candranugraha yang juga sebagai supervisi program *internship* pada tim *Payment.* Pada bagian *frontend* pada tim *Payment* dipimpin oleh Bapak Susetyo Aribowo yang menjadi pembimbing lapangan dan memberikan tugas serta melaporkan hasil pekerjaan yang sudah dilakukan selama proses kerja magang. Dalam tim *Payment* terdapat beberapa posisi berbeda dengan *job desk* masing-masing saling berkaitan dalam pengembangan fitur pada tim *Payment* sebagai berikut.

- *Technical Lead* sebagai pemimpin pada tim dan memberikan arahan terhadap tugas-tugas yang sedang dikerjakan oleh setiap anggota tim.
- *Scrum Master* yang bertugas mengatur *software development life cycle* (SDLC) dalam tim.
- *Product Manager* bertugas mengatur tugas-tugas yang akan dikerjakan oleh tim dalam suatu *sprint* dan membuat prioritas tentang tugas-tugas yang harus segera diselesaikan. *Product Manager* juga bertugas menjembatani antara tim *developer* dan tim *business* maupun *UI*/*UX*.

- *Quality Assurance* bertugas menguji program yang telah dikembangkan untuk menemukan kesalahan atau *bug* pada program agar program dapat berjalan tanpa masalah berarti.
- *Backend Engineer* bertugas mengelola data pada basis data, membuat *API* ataupun *endpoint* untuk diakses oleh *Frontend Engineer*.
- *Frontend Engineer* bertugas mengolah data dari *restAPI* yang telah dibuat oleh *Backend Engineer* untuk mengembangkan fitur *Payment* sesuai dengan rancangan antarmuka yang telah dibuat

Agile scrum adalah nama *framework* yang digunakan oleh tiket.com sebagai alat manajemen pengembangan piranti lunak, karena bisa digunakan untuk proyek berskala besar. Koordinasi yang diatur dalam metode *scrum*, ada sebuah *list* dari semua tugas yang akan diberikan kepada tim *payment* bernama *backlog* yang diatur oleh *Product Manager.* Kemudian dari *backlog* akan dibagikan tugas-tugas tersebut kepada masing-masing developer pada rapat yang bernama *Sprint Planning* yang dilakukan dua minggu sekali, dan selama dua minggu disebut dengan istilah satu *sprint.* Masing-masing tugas memiliki bobot untuk mengevaluasi hasil kerja tim. Kemudian ada juga yang dinamakan *weekly sync-up* yaitu rapat koordinasi yang dilakukan tim *Payment* setiap minggu untuk berbagi progress dan kendala yang dihadapi dalam mengerjakan tugas yang sudah diberikan.

Pekerjaan seara teknis menggunakan *git workflow* dalam pengerjaan tugas yang diberikan. *Git flow* yang dilakukan adalah membuat *repository* baru saat memulai *project* baru tetapi tugas yang diberikan bisa saja hanya fitur baru pada *project* yang sudah ada. Jika *project* sudah ada maka pertama kali adalah melakukan *clonning* terhadap *repository* tersebut, lalu jika melakukan perubahan segera

melakukan *git commit* yaitu menyimpan suatu perubahan dilengkapi dengan pesan singkat mengenai melakukan apasaja dalam perubahan tersebut, kemudian git *push* untuk mengirim perubahaan yang sudah disimpan dalam *commit* ke *server* github agar dapat dilihat oleh semua *developer* yang bekerja dalam repository yang sama. *Developer* lain yang akan melanjutkan harus melakukan *git pull* terlebih dahulu yaitu menyimpan perubahaan dalam *commit* yang sudah ada sampai saat ini ke dalam repository lokal mereka.

Pada umumnya *repository* dibagi menjadi beberapa *branch* untuk keperluan *versioning* yang lebih baik, kemudian developer melakukan *pull request* terlebih dahulu sebelum melakukan *git merge* atau menggabungkan perubahan ke *branch* utama, yaitu menunggu *review* dan persetujuan supervisi untuk kemudian digabungkan dengan branch utama menjadi perubahan yang sudah valid.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama proses kerja magang di PT Global Tiket Network sebagai *frontend enginer intern* adalah mengembangkan *website* aplikasi *dashboard internal* yang berfungsi untuk mengatur dan generate kode promo ataupun *discount* yang akan diadakan oleh divisi bisnis atau komersil. Bahasa yang digunakan pada bagian *frontend* adalah javascript dengan *framework* ReactJS yang menggunakan representasi data JSON untuk berkomunikasi dengan *endpoint* yang telah disediakan tim *backend* untuk kemudian dilakukan *query* pada data JSON yang didapatkan menggunakan *GraphQL* untuk mengolah data yang dibutuhkan saja pada masing-masing komponen atau untuk mengaplikasikan fungsi CRUD yang sudah disediakan APInya.

3.3 Proses Pelaksanaan

3.3.1. Proses Pelaksanaan

Pada bagian ini akan dijelaskan apa saja yang dilakukan selama menjalani proses kerja magang, realisasi kerja magang dapat dilihat pada Tabel 3.1 berikut.

Tabel 3.1. Realisasi Kerja Magang

Minggu ke-	Pekerjaan yang dilakukan
1	• Mempelajari framework ReactJS dan GraphQL • Mempelajari React lifecycle
2	• Membuat type, model, dan query / mutation GraphQL untuk melakukan CRUD fitur penambahan promo code pada *dashboard*
3	• Menambahkan fitur CRUD pada sub menu lainnya pada *dashboard*
4	• Melakukan *checking* pada fitur-fitur yang ada dan memperbaikinya jika belum sesuai dengan ketentuan
5	• Membuat type, model, dan query / mutation GraphQL untuk fitur CRUD pada bagian menu yang belum ditambahkan fitur CRUD
6	• Melakukan testing pada fitur-fitur yang selesai dibuat • Mempelajari EcmaScript 6 khususnya. Untuk *destructuring* data yang diperoleh dari *endpoint* •
7	• Membuat komponen lebih dinamis agar tidak error ketika menampilkan data kode promo lama yang memiliki kekurangan value pada field objectnya, serta tetap menampilkan data yang lengkap pada data baru

Tabel 3.1. Realisasi Kerja Magang (lanjutan)

Minggu ke-	Pekerjaan yang dilakukan
8	• Membuat fitur edit data promo code yang butuh menghubungkan data lebih dari satu endpoint yang berbeda untuk kemudian dicocokan berdasarkan parameter opsi yang dipilih • Melakukan *destructuring* pada data JSON yang didapat karena memiliki struktur kedalaman yang berbeda-beda untuk kemudian mendapatkan data yang dibutuhkan dari beberapa *endpoint* agar dapat kemudian dicocokan
9	• Menyelesaikan tugas sebelumnya dan melakukan testing pada fitur-fitur lain yang telah diselesaikan

Selama melakukan proses kerja magang di PT.Global Tiket Network, dipinjamkan sebuah laptop yang menggunakan *Operating System* (OS) berbasis Linux yaitu distro Ubuntu. Minggu pertama selain belajar juga mempersiapkan laptop agar sesuai dengan *environtment* dengan menginstall library dan piranti lunak yang akan digunakan. Mempelajari ReactJS dari dokumentasi resminya dan mulai dengan mencoba membuat *game* "tic tac toe" yang disediakan tutorialnya di website dokumentasi react. Dalam game tersebut memberikan contoh bagaimana komunikasi data antar komponen menggunakan sesuatu yang disebut dengan "props" dalam ReactJS. Lalu memahami *lifecycle* yang ada dalam ReactJS yang pada intinya terdiri dari tiga tahap yaitu *Initialization, Update,* dan *Destruction*.

Kemudian minggu kedua mulai belajar membuat komponen untuk menarik data dari *endpoint* menggunakan GraphQL yaitu *type, model,* dan *query/mutation*.

Pertama membuat *type* terlebih dahulu yaitu komponen yang berisi susunan tipe data yang akan ditampung ketika mendapat data dari atau mengembalikan data dalam format JSON. Kemudian membuat *Query* atau *Mutation*, keduanya adalah komponen yang akan mengambil data dari *endpoint* yang diberikan sehingga berbentuk struktur Object yang kedalamanya akan menyesuaikan dengan data yang dikembalikan dari *endpoint.* Perbedaanya terdapat dalam fungsinya, *Query* ditujukan untuk menarik data atau dengan istilah *fetch* data, sedangkan *Mutation* untuk melakukan perubahan data seperti *create/write*, *update*, dan *delete* data. Terakhir *Model* adalah komponen yang merupakan konfigurasi dalam penarikan data, seperti url *endpoint* yang digunakan, metode yang digunakan *Query* atau *Mutation*.

Untuk membuat metode *Query/Mutation* dapat melihat bentuk data yang dikembalikan oleh *endpoint* dalam aplikasi yang bernama SwaggerAPI, yaitu sebuah *tools* yang membantu membuat dokumentasi API serta dilengkapi sebuah aplikasi web yang dapat mempermudah pengguna dalam hal ini *frontend engineer* untuk memahami struktur data yang dibalikan dan dapat mencoba melakukan request penarikan data langsung di halaman tersebut (SwaggerAPI, 2019). GraphQL juga memiliki halaman *web* yang berjalan di *localhost* untuk mencoba apakah komponen yang sudah dibuat dapat berfungsi, tanpa perlu disambungkan dahulu ke web yang akan dikerjakan. Komponen-komponen yang dibutuhkan sudah siap lalu dapat diaplikasikan untuk membuat fitur CRUD pada halaman web dengan menggunakan *model* yang sudah dibuat. Tugas yang dilakukan dari minggu ke-2 hingga minggu ke-5 adalah melengkapi semua menu dan sub menu pada dashboard *Discount Engine* dengan fitur CRUD.

Tugas yang dilakukan dari minggu ke-6 hingga minggu terakhir berfokus pada melengkapi komponen ReactJS pada bagian *create*. Pada bagian *create* terdapat banyak *form* dan terdiri dari *text input, radio option*, maupun menu dropdown yang mendapatkan isi dropdownnya dari API. Salah satu yang diperhatikan adalah form *validation* sesuai ketentuan yang diberikan oleh *product manager*, agar tidak terjadi kesalahan *input* oleh *user* yang akan menggunakan *dashboard* tersebut. *Form validation* dibuat dengan membuat komponen atau fungsi ReactJS khusus hanya untuk memeriksa *value* dari *text input* kemudian kemudian diberikan kondisi sesuai ketentuan yang diberikan oleh *product manager*, jika tidak sesuai dengan kondisi tersebut maka komponen akan mengembalikan pesan *error* sesuai dengan yang sudah diatur pada masing-masing kondisi.

Kemudian mempelajari fungsi *map*, *filter*, *arrow function*, dll yang terdapat pada EcmaScript 6 untuk membuat fitur dropwdown yang opsi pilihannya didapatkan dari *endpoint* yang diberikan oleh tim *backend.* Fungsi *map* dan *filter* digunakan untuk memperoleh dan mengolah data yang merupakan bentuk kembalian dari *endpoint* dalam representasi JSON dan akan di *assign* ke *variable* baru. *Map* digunakan untuk mengakses suatu *array of object* lalu dapat mengembalikan *value* dalam bentuk *array* atau *array of object* kembali dengan melakukan perubahan *value* atau untuk mengembalikan *key* yang diperlukan saja. *Filter* dapat mengakses suatu *array of object* lalu ditujukan untuk mencari data dengan *value* tertentu, kemudian mengembalikan sebuah *array of object* sesuai kondisi. Terakhir adalah *Arrow Function,* yaitu sebuah alternatif dari function regular, perbedaanya *Arrow Function* memiliki bentuk yang lebih singkat dan fitur

bindings yang mempermudah dalam mengolah data bahkan untuk *destructuring* (MDN Mozilla, 2020).

3.3.2. Tools yang Digunakan

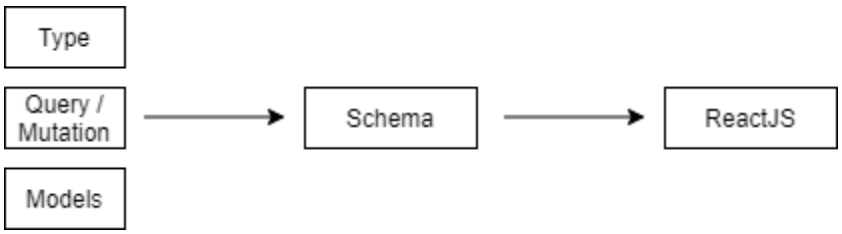
Hari pertama dipinjamkan laptop oleh kantor dengan *Operating System* (OS) Linux Ubuntu, kemudian melakukan *setup* untuk menyesuaikan dengan *environtment* yang digunakan dan piranti lunak yang akan digunakan, yaitu installasi NodeJs sebagai *framework* dasar yang akan digunakan dan NPM sebagai *package manager* untuk NodeJs, serta NVM yang digunakan untuk menyesuaikan versi dari NodeJs dan NPM yang digunakan. Selanjutnya menggunakan ReactJS untuk mengerjakan tugas yang akan diberikan sebagai *frontend engineer* dan installasi Webstorm dari InteliJ IDEA sebagai *IDE* yang digunakan untuk melakukan pemrograman javascript dengan *framework* ReactJS.

Ada beberapa *tools* yang digunakan dalam bekerja dan berkoordinasi dalam tim dan untuk keperluan *deployment*. Slack adalah sebuah piranti lunak yang digunakan untuk berkomunikasi serta dapat berbagi file dalam tim bahkan antar tim dalam bentuk komunikasi yang tidak terlalu formal, Slack memiliki *channel* diskusi yang terpisah-pisah sehingga memudahkan dalam berdiskusi dengan siapa saja pada topik apa saja, dan juga terorganisir dengan baik. *Platform* yang digunakan untuk memantau versi (*versioning*) dan berbagi *source code* atau biasa dikenal dengan *git* adalah Github. Pada umumnya setiap project besar memiliki *repository,* yang terdiri dari *branch* master yang digunakan sebagai fitur yang sudah *release* , *branch* stagging digunakan untuk keperluan *testing* dan *branch* masing-masing fitur. Pada saat pengerjaan fitur setelah *commit* dan *push* ke *local branch* untuk

kemudian diperiksa dan di review oleh supervisi akan dilakukan *pull request* terhadap *branch* yang ingin di gabungkan (*merge*) seperti *stagging* atau bahkan *master.*

Fitur yang siap *release* dan sudah melalui *testing* kemudian akan ditangani oleh bagian Infra yang bertanggung jawab atas *resource* seperti *server* dan *database* untuk dilakukan *deployment*. Developer akan menandai fitur yang siap release dengan git tag, kemudian git tag dapat diakses oleh piranti lunak bernama Jenskin. Jenskin sebuah *tools* untuk mengintegrasikan kode dari *shared repository* yang dalam hal ini adalah Github secara berulang-ulang, sehingga tidak perlu konfigurasi *server* dan lainnya untuk *deployment* setiap kali fitur *release* (Abdurohman, 2017).

3.3.3. Implementasi



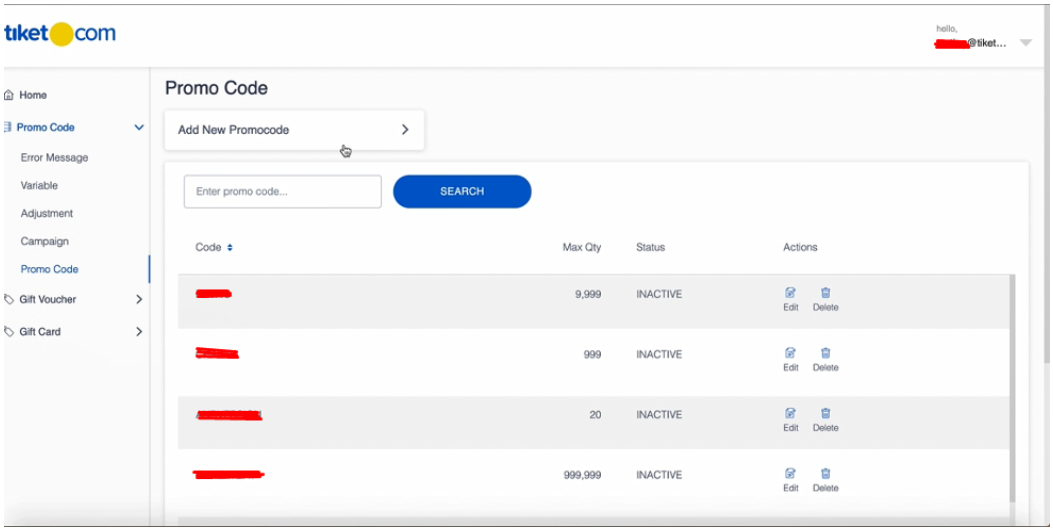
Gambar 3.1. Skema GraphQL dan ReactJS

Pertama lakukan penarikan data dari endpoint yang sudah disediakan dan dapat melihat dokumentasinya dari halaman lokal Swagger yang sudah dibuatkan tim *backend.* Lalu untuk menarik data menggunakna GraphQL memerlukan komponen yang sudah dijelaskan sebelumnya yaitu *Type, Query / Mutation,* dan *Models*. Setelah konfigurasi yang dilakukan pada *Models* kemudian dapat mencoba menampilkan data dengan menggunakan halaman lokal bernama GraphiQL, GraphiQL berfungsi sebagai alat *debugging* untuk mencoba membuat GraphQL *schema* apakah sudah berfungsi, dan apakah komponen-komponen sebelumnya

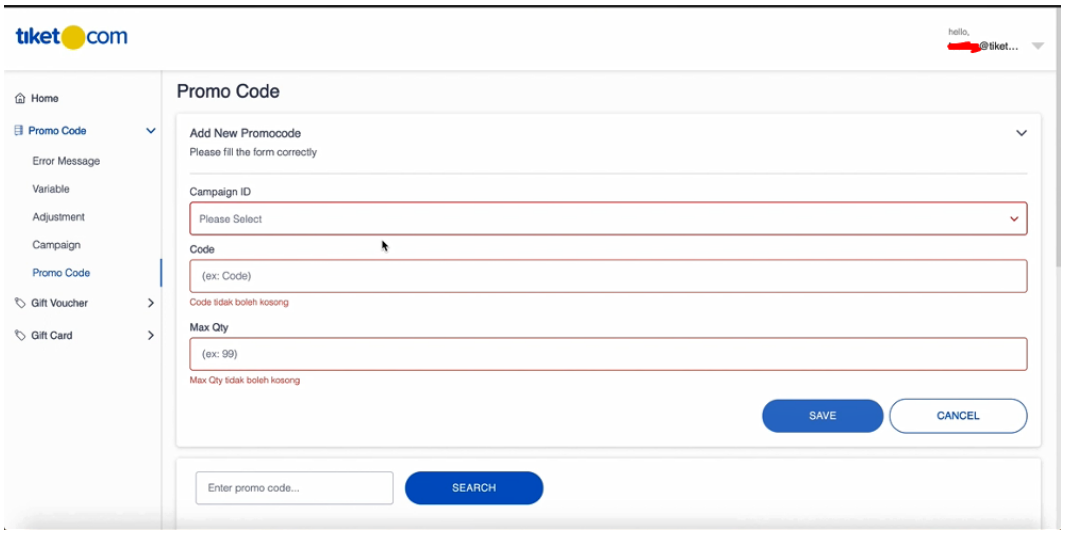
sudah bekerja. GraphQL *schema* adalah sebuah kueri yang berbentuk menyerupai struktur JSON, yang kemudian data yang dikembalikan dari *endpoint* akan mengikuti struktur *schema,* akan tetapi *schema* yang dibuat juga harus mengikuti bentuk data yang disediakan.

Selanjutnya Schema yang berjalan dengan baik dan sesuai dapat disimpan dalam file js yang selanjutnya akan di *export* menggunakan Export Default, fitur dari JavaScript generasi terbaru yaitu EcmaScript 6. Pada file ReactJS kemudian dapat memanggil *schema* menggunakan bantuan *Apollo Client*, yaitu salah satu library yang dapat menghubungkan ReactJS dengan GraphQL. *Schema* yang telah di *import* dapat digunakan seperti melakukan API *Request* pada umumnya. Selanjutnya hanya tinggal membuat halaman yang berfungsi sebagai *interface* agar dapat dengan mudah melakukan CRUD dan pengolahan data pada sisi *Frontend*-nya.

Pada halaman *dashboard internal* Discount Engine ini, setiap menu memiliki fungsi CRUD pada bagian atas terdapat form untuk *create* data baru maupun edit data akan menggunakan form yang sama hanya saja *method* yang dipakai akan berbeda. Form akan muncul ketika tombol "*add new PromoCode*" ditekan, dapat dilihat pada Gambar 3.2 dan Gambar 3.3

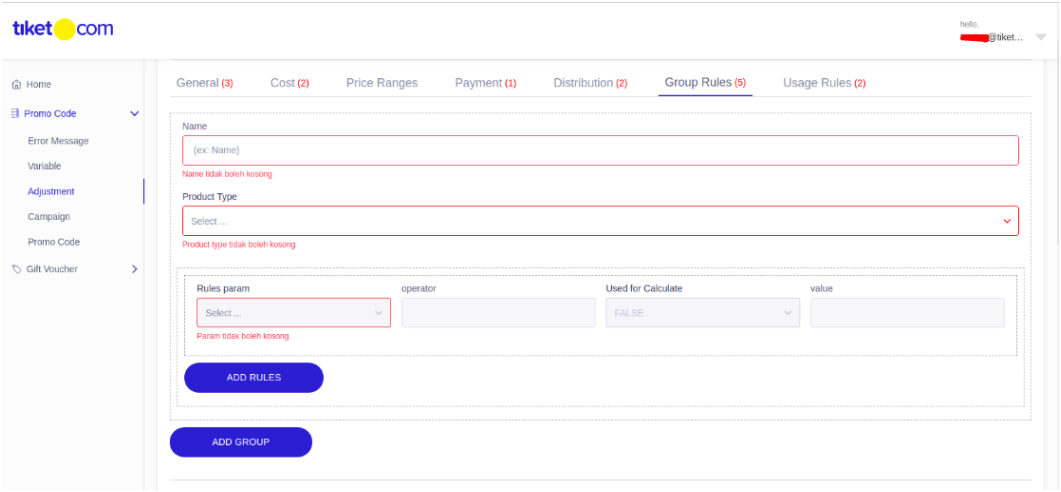


Gambar 3.2. Halaman Menu PromoCode

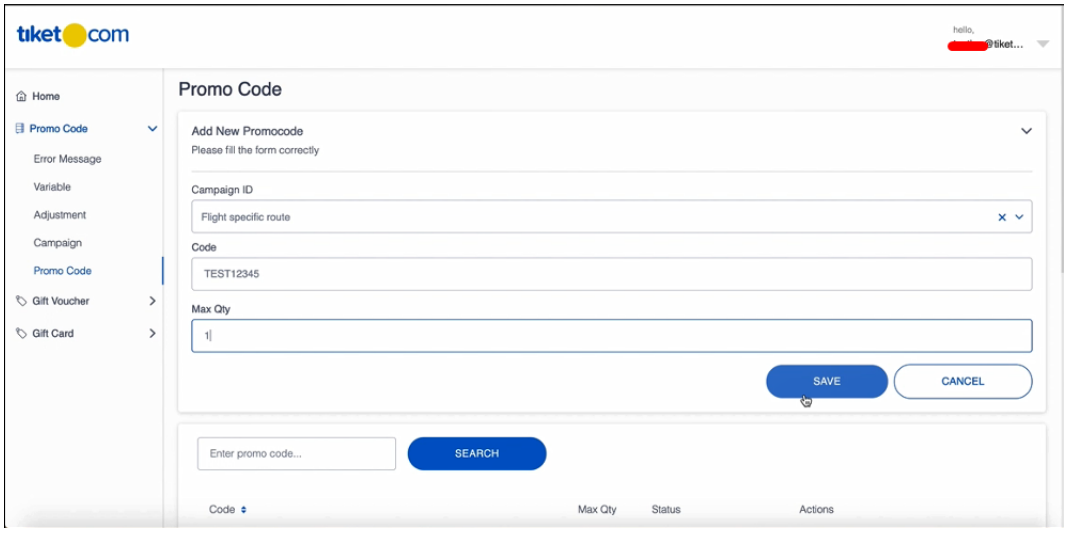


Gambar 3.3. Form Tidak Terisi

Pada Gambar 3.3 terdapat *outline* merah di *text input* karena fungsi validasi bekerja yaitu form belum terisi tetapi sudah ditekan tombol *save*, sehingga form tidak akan mengirim data menggunakan API, tetapi memberi notifikasi kepada *user* bahwa masih ada form yang kosong. Ada juga halaman menu lain yang memiliki banyak sub menu menampilkan notifikasi form belum terisi seperti pada Gambar 3.4. Angka yang muncul disamping judul sub menu berarti menandakan ada berapa *form input* yang wajib diisi tetapi masih kosong saat ditekan tombol *save.*

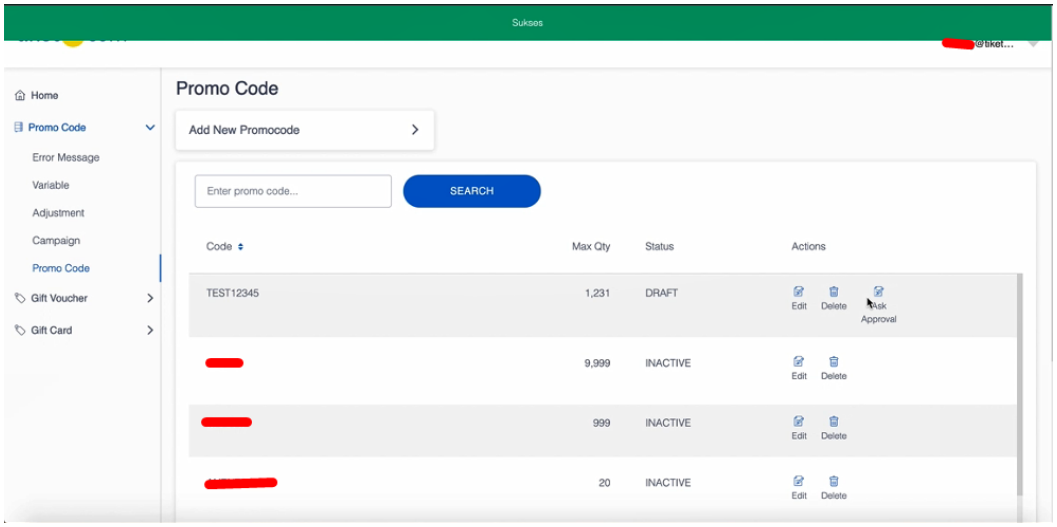


Gambar 3.4. Notifikasi Validasi Form



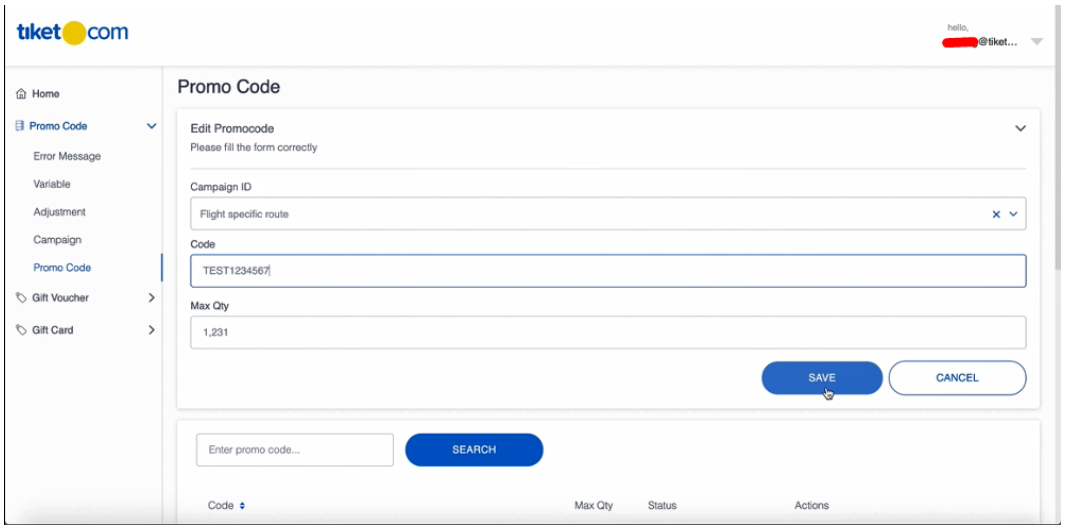
Gambar 3.5. Menambahkan Data Baru

Selanjutnya jika dicoba untuk memasukan data baru dapat dilihat pada Gambar 3.5 setelah tombol *save* di tekan, data baru akan dikirim menggunakan API lalu kemudian akan me-*trigger* bagian *render* data dibawahnya untuk *refetch query* kembali agar dapat langsung menampilkan data yang baru saja dimasukan. Setelah sukses terkirim dan data tampil bersamaan juga akan muncuk sebuah komponen *toast* dari atas halaman yang memberi tahu bahwa aksi yang dilakukan sukses, seperti pada Gambar 3.6.



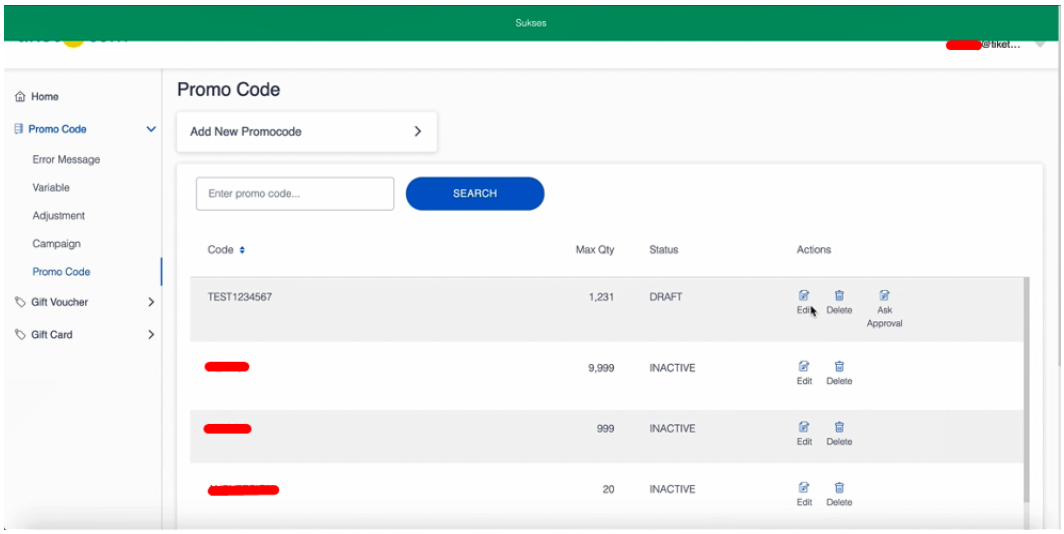
Gambar 3.6. *Toast* dan Data Baru Ditampilkan

Lalu dibagian yang melakukan *render* data yang didapat dari *endpoint* terdapat sebuah kolom yang berisi aksi yang dapat dilakukan yaitu dengan *method update* dan *delete*. Contohnya pada Gambar 3.7 akan mengubah *value* data yang baru saja dibuat dengan menekan tombol edit. Saat tombol edit ditekan, data tersebut akan mengirimkan data id dari data tersebut ke komponen form yang ada pada bagian atas.



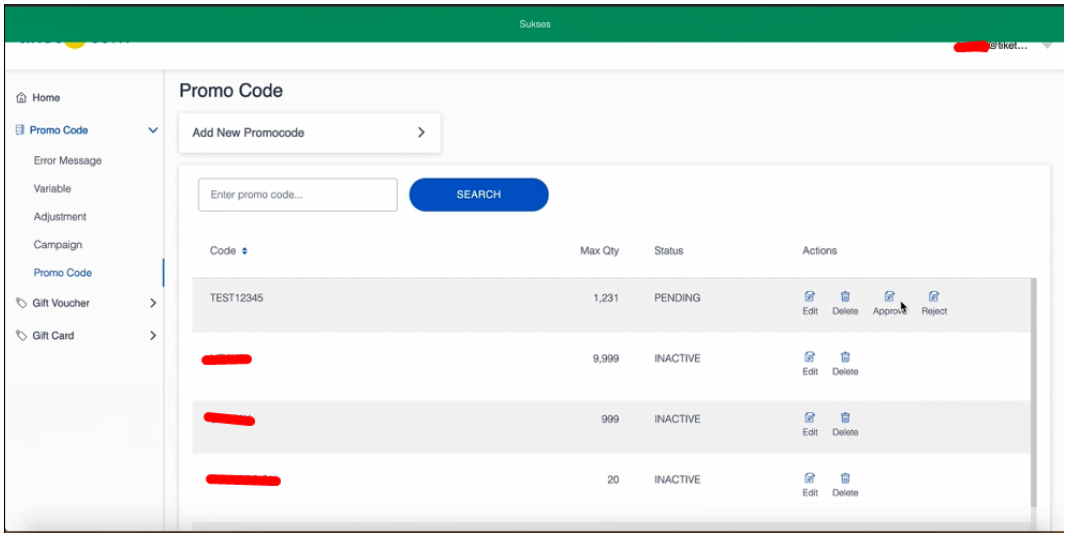
Gambar 3.7. Form Edit Data

Form yang sebelumnya digunakan untuk membuat data baru akan otomatis terisi dengan data dari id yang dikirimkan dari komponen *render* data. Sehingga *user* dapat mengubah *value* dari data tersebut langsung di form yang telah otomatis terisi. Kemudian saat tombol save ditekan form akan mengirimkan data dengan metode *update* melalui API, dan bagian *render* data akan *refetch* kembali untuk menampilkan data terbaru yang baru saja dirubah seperti pada Gambar 3.8.



Gambar 3.8. Data Telah Diubah dan Ditampilkan

Pada data yang statusnya tidak "INACTIVE" akan terdapat tombol aksi untuk melakukan *method update* contohnya pada status "DRAFT" akan terdapat aksi "Ask Approval" yang berarti untuk menaikan status dari "DRAFT" menjadi "PENDING", dapat dilihat pada Gambar 3.8 terdapat tombol aksi "Ask Approval". Kemudian pada kondisi status "PENDING" akan muncul dua aksi yaitu "Approve" untuk merubah statusnya menjadi "ACTIVE" dan tombol "Reject" yang akan merubah statusnya menjadi "PENDING", seperti pada Gambar 3.9.



Gambar 3.9. Data Dengan Status Pending

Data dengan status "INACTIVE" dapat diaktifkan kembali dengan menekan tombol edit dan *save,* kemudian data akan berada dalam status "DRAFT" kembali.

3.4 Kendala yang ditemukan

Kendala teknis yang dihadapi pertama kali adalah kurangnya pengetahuan akan *framework* ReactJS dan GraphQL yang digunakan selama pelaksanaan kerja magang. Selama pelaksanaan kerjan magang juga mengahadapi masalah dalam tools Github, yaitu sering kali salah mengambil *branch* sebagai sebuah dasar dari *branch* baru atau dalam satu *branch* tercampur dengan fitur lain sehingga tidak bersih dan bisa menimbulkan masalah.

Kendala non-teknis yang dihadapi adalah sedikit canggung untuk berbicara kepada anggota tim yang lebih tua dan baru kenal, serta malu karena banyak bertanya dan meminta bantuan saat terjadi *error* pada ReactJS maupun GraphQL.

3.5 Solusi atas kendala yang ditemukan

Solusi yang didapatkan untuk menghadapi kendala teknis dengan bertanya kepada teman satu tim dan pembimbing lapangan, banyak membaca blog seperti Medium dan dokumentasi resmi masing-masing framework serta melihat *pull request* yang masih ada maupun yang sudah di-*merge*. Pembimbing lapangan juga beberapa kali mengajarkan bagaimana *branching* yang baik agar *branch* yang digunakan bersih hanya fitur yang sedang dikerjakan saja dan tidak mengganggu fitur lain.

Solusi yang didapatkan untuk masalah non-teknis yaitu belajar berkomunikasi lebih baik lagi dan sering berkomunikasi dengan orang yang baru kenal, menghindari rasa canggung tetapi tetap sopan. Pembimbing lapangan mengajari beberapa cara untuk melakukan *debugging* ketika ada *error* dan bagaimana mengatasinya sesuai *error* yang dihadapi, serta *keyword* yang dapat digunakan untuk belajar melalui internet. Sehingga tidak selanjutnya jarang bertanya kecuali diajari hal baru atau saat proses *review pull request*.